This report is based on studies performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology. The work was sponsored by the Strategic Defense Initiative Organization under Air Force Contract F19628-90-C-0002.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER

Hugh L. Southall

Hugh L. Southall, Lt. Col., USAF
Chief, ESD Lincoln Laboratory Project Office

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LINCOLN LABORATORY

A MODIFIED FAST FOURIER TRANSFORM

A.L. STEINBACH
C.H. WANNER
Group 35

TECHNICAL REPORT 867

24 JANUARY 1990

LEXINGTON MASSACHUSETTS

ABSTRACT

This report presents a derivation of the modified discrete Fourier transform, which has the property that the origin in frequency space appears in the center of the plot rather than at the edges (one dimension) or at the four corners (two dimensions), as in conventional treatments. Also included is a listing of an unusual fast Fourier transform (FFT) program for calculating the two-dimensional, modified discrete Fourier transform. The program makes use of the Eklundh fast matrix transposition algorithm and can transform arrays that are much too large to fit within the internal memory of the computer. By way of example, a complex array of size 2048 $\times$ 2048 can easily be transformed on a microcomputer with a 40-MB hard disk.

ACKNOWLEDGMENTS

We thank Kent Edwards and Charles Meins for their helpful suggestions and comments and especially for their unfailing support during the course of this work. We also thank Marion Becker for her precise and careful preparation of this manuscript.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

1. INTRODUCTION AND SUMMARY

In radar theory, one frequently needs to compute Fourier transforms or quantities, such as convolutions and cross-correlations, which are most easily calculated in terms of such transforms. The required quantity is usually either a one- or two-dimensional continuous Fourier transform, which is subsequently reduced to a discrete form suitable for numerical calculation. In standard treatments[1,2,3] of the one-dimensional case, computation of the forward and inverse continuous Fourier transforms,

$$F(u) = FT\{f(x)\} = \int_{-\infty}^{\infty} f(x)e^{-i2\pi ux}dx \tag{1a}$$

$$f(x) = FT^{-1}\{F(u)\} = \int_{-\infty}^{\infty} F(u)e^{i2\pi ux}du \quad , \tag{1b}$$

is reduced to computation of the corresponding forward and inverse discrete Fourier transforms (DFT and IDFT, respectively):

$$F(u_m) = DFT\{f(x)\} = \frac{1}{N}\sum_{k=0}^{N-1} f(x_k)e^{-i(2\pi/N)mk} \tag{2a}$$

$$f(x_k) = IDFT\{F(u)\} = \sum_{m=0}^{N-1} F(u_m)e^{i(2\pi/N)km} \quad , \tag{2b}$$

where

$$x_k = k\Delta x = k\,\frac{X}{N}\ ;\ k = 0, 1, \ldots, N-1$$

$$u_m = m\Delta u = \frac{m}{X}\ ;\ m = 0, 1, \ldots, N-1 \tag{3}$$

and

X = length of basic integration interval
N = number of sampling points
Δx and Δu are the sampling distances in direct and inverse (i.e., frequency) space, respectively.

In deriving Eq. (2a) from Eq. (1a), one in effect converts an integral over the symmetric interval $[-X/2, X/2]$ into one over the asymmetric interval $[0, X]$. By choosing the sampling points in direct space and frequency space (x_k and u_m, respectively) to be those in Eq. (3) above, one obtains an especially simple expression for the DFT. An equally simple expression is likewise found for the IDFT. The above formulation has, however, one very significant disadvantage. Although one apparently calculates the Fourier spectrum, $F(u_m)$, within the frequency interval $[0, U]$, one obtains, in fact, the spectrum over the interval [-U/2, U/2] with the negative frequencies displaced so as to occupy the interval [U/2, U].[3,4] Thus, when one calculates the DFT of a one-dimensional array, one finds that the origin of frequency space in the transformed array appears not at the center (as one would prefer) but rather at the two ends. If the one-dimensional formalism of Eq. (2) is then extended to the two-dimensional case, as in Reference 2, one finds that the result of applying a two-dimensional DFT to a two-dimensional input array is a Fourier transform array whose origin (in frequency space) lies not in the center of the output picture but rather at the four corners. When one attempts to use such one- and two-dimensional DFTs and IDFTs to perform convolutions and cross-correlations, the situation becomes even more confusing.

In this report, we present an alternative derivation of the discrete Fourier transform which avoids most of the needless complexity of previous treatments and the confusion which follows therefrom. In addition, we describe and include FORTRAN listings of four programs. These include a two-dimensional fast Fourier transform (FFT) program, which not only implements the modified DFT and IDFT to be described below, but also has the interesting property that it can be used to transform arrays which are much too large to fit within the internal memory of the computer. The program makes use of the Eklundh fast matrix transposition algorithm,[5] for which, to the best of our knowledge, no FORTRAN translation has ever been published. Such a two-dimensional FFT makes it possible to Fourier transform even rather large arrays — say 2048 $\times$ 2048 complex — on a microcomputer, provided only that the attached hard disk has space sufficient to store the entire input (output) array. Recently, Serzu and Moon[6] described their FORTRAN translation of the Anderson algorithm;[7] this program also permits Fourier transformation of very large two-dimensional arrays. However, the program described in the present report is simpler conceptually and requires approximately 170 fewer lines of FORTRAN to code. Furthermore, according to Anderson, the method described below "is as efficient as the method of this [Anderson's] paper, and may be desirable if special purpose FFT hardware is to be employed."

The remainder of this report is organized as follows. In Section 2 we derive the modified one-dimensional DFT and IDFT from the corresponding continuous transforms. These modified quantities have the property that the basic sampled interval is placed symmetrically about zero in both direct and inverse (transformed) space. The straightforward calculation which follows shows that the modified discrete transforms are exact inverses of one another, just as are the conventionally defined quantities [see Eq. (2)]. Finally, we demonstrate the proper use of the one-dimensional modified DFT and IDFT in computing a convolution which is free of wraparound error. In Section 3 we extend the results of Section 2 to the two-dimensional case, with particular emphasis being placed on a description of the operation of the two-dimensional FFT program, FFT2D. These results are followed, in Section 4, by a brief summary of the programs included and the mathematical quantities they calculate. A complete listing of all programs discussed follows in the Appendix.

2. ONE-DIMENSIONAL FOURIER TRANSFORMS

The forward and inverse Fourier transforms are given by Eq. (1). To obtain the continuous Fourier transform in terms of the modified DFT, we proceed as follows:

Write the Fourier integral over the infinite interval as a sum of integrals over consecutive finite intervals of length X, and let the basic interval coincide with [-X/2, X/2]. Choose X large enough to ensure that the integral over the basic interval is already a good approximation to the infinite integral. We obtain

$$F(u) = \int_{-\infty}^{\infty} f(x)e^{-i2\pi ux}dx$$

$$= \sum_{m'=-\infty}^{\infty} \int_{(m'-1/2)X}^{(m'+1/2)X} f(x)e^{-i2\pi ux}dx \quad .$$

Now setting $x' = x - m'X$,

$$F(u) = \sum_{m'=-\infty}^{\infty} \int_{-X/2}^{X/2} f(x' + m'X)e^{-i2\pi u(x'+m'X)}dx'$$

$$= \int_{-X/2}^{X/2} \left[\sum_{m'=-\infty}^{\infty} f(x + m'X)e^{-i2\pi um'X}\right]e^{-i2\pi ux}dx \quad . \tag{4}$$

Next, evaluate F(u) on the evenly spaced set of points

$$u_m = [m - (M - 1)/2]\Delta u = [m - (M - 1)/2]/X$$

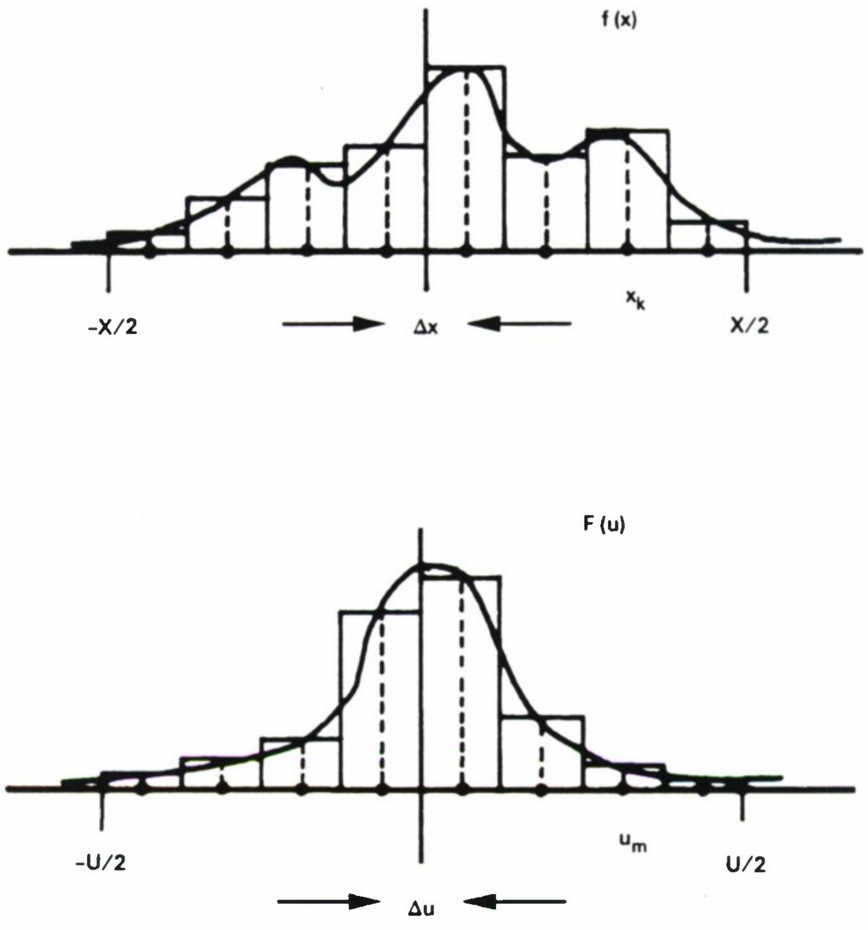
where

M = sufficiently large even integer (determined below)

$m = 0,1,...,M-1$

$U = M\Delta u = M/X$ (see Figure 1).

Choose U large enough to ensure that F(u) is essentially zero outside the interval [-U/2, U/2]. This is accomplished by choosing M sufficiently large. Note carefully that the distance between adjacent sample points in frequency space is suggested by the Sampling Theorem:[8] If f(x) is identically zero outside the interval [-X/2, X/2], then it is possible to recover F(u) arbitrarily accurately in terms of samples spaced at intervals $\Delta u = 1/X$:



$X = N\Delta x$ $\qquad$ $U = N\Delta u$

$\Delta x = 1/U$ $\qquad$ $\Delta u = 1/X$

$x_k = (k - \frac{N-1}{2})\,\Delta x$ $\qquad$ $u_m = (m - \frac{N-1}{2})\,\Delta u$

where $k = 0,1,\ldots,N-1$ $\qquad$ where $m = 0,1,\ldots, N-1$

129087-1

Figure 1. Calculation of the forward and inverse Fourier transforms in one dimension.

$$F(u_m) = \int_{-\infty}^{\infty} f(x)e^{-i2\pi u_m x}dx$$

$$= \int_{-X/2}^{X/2} [\underbrace{\sum_{m'=-\infty}^{\infty} f(x + m'X)e^{-i2\pi u_m m'X}}_{f_p(x)}]e^{-i2\pi u_m x}dx$$

$$f_p(x) = \sum_{m'=-\infty}^{\infty} f(x + m'X)e^{-i2\pi[m - (M - 1)/2](1/X)m'X}$$

$$= \sum_{m'=-\infty}^{\infty} f(x + m'X)e^{-i2\pi mm'}e^{i\pi(M - 1)m'}$$

$$= \sum_{m'=-\infty}^{\infty} (-1)^{m'}f(x + m'X) \quad .$$

We then obtain

$$F(u_m) = \int_{-\infty}^{\infty} f(x)e^{-i2\pi u_m x}dx$$

$$= \int_{-X/2}^{X/2} [\underbrace{\sum_{m'=-\infty}^{\infty} (-1)^{m'}f(x + m'X)}_{f_p(x)}]e^{-i2\pi u_m x}dx \tag{5}$$

where

$$u_m = (m - \frac{M - 1}{2})/X$$

M = even integer

$m = 0,1...,M - 1$.

The above result is exact. Note that $f_p(x) = f(x)$ for $x \in [-X/2, X/2]$ if and only if $f(x)$ vanishes outside this interval. If $f(x)$ is non-zero outside this interval, then calculating $F(u_m)$ by using a truncated approximation to $f_p(x)$ is equivalent to truncating the defining integral. This resulting truncation error is usually referred to in the literature as aliasing in the time domain[1] (i.e., direct space).

As before, assume that X is chosen large enough that f(x) is effectively zero outside the interval [-X/2, X/2]. Accordingly, approximate $f_p(x)$ by the n = 0 term in its expansion; i.e., set $f_p(x) = f(x)$. Then calculate $F(u_m)$ approximately using the midpoint rectangle rule (see Figure 1):

$$\begin{aligned} F(u_m) &= \int_{-X/2}^{X/2} f_p(x)e^{-i2\pi u_m x}dx \\ &= \int_{-X/2}^{X/2} f(x)e^{-i2\pi u_m x}dx \\ &= \sum_{k=0}^{N-1} f(x_k)e^{-i2\pi u_m x_k}\Delta x \end{aligned} \tag{6}$$

where

$$x_k = (k - \frac{N-1}{2})\Delta x \text{ and } X = N\Delta x \quad .$$

We are immediately confronted by the problem of properly choosing Δx, or equivalently N, since $N = X/\Delta x$. According to the Sampling Theorem, if F(u) is zero outside the interval [-U/2, U/2], then it is possible to recover f(x) arbitrarily accurately using samples spaced at intervals $\Delta x = 1/U$. To obtain a high quality approximation to the above integral using the midpoint rectangle rule, one therefore chooses the rectangle width $\Delta x = 1/U$. The non-zero rectangle width causes an error in the calculation of $F(u_m)$; this error is usually called aliasing in the frequency domain.[9] Assuming now that F(u) is essentially zero outside the interval [-U/2, U/2] and $\Delta x = 1/U$, the foregoing arguments give

$$\left.\begin{aligned} M &= U/\Delta u = UX \\ N &= X/\Delta x = XU \end{aligned}\right\} \text{ which implies that } M = N \quad .$$

Using M = N in the summation in Eq. (6) gives

$$F(u_m) \approx \sum_{k=0}^{N-1} f(x_k)e^{-i2\pi u_m x_k}\Delta x \equiv F_a(u_m) \tag{7}$$

$$\begin{aligned} u_m x_k &= \frac{(m - \frac{N-1}{2})}{X} \cdot (k - \frac{N-1}{2})\Delta x \\ &= \frac{(m-c)}{N\Delta x} \cdot (k-c)\Delta x \\ &= (1/N)(m-c)(k-c) \\ &= (1/N)(c^2 - mc - kc + mk) \quad . \end{aligned}$$

Inserting into Eq. (7), we have

$$F_a(u_m) = e^{-i(2\pi/N)(c^2 - mc)}\Delta x \cdot \sum_{k=0}^{N-1} f(x_k)e^{-i(2\pi/N)(mk - kc)}$$

$$= e^{-i(2\pi/N)(c^2 - mc)} \cdot X \cdot \left\{(1/N) \sum_{k=0}^{N-1} f'(x_k)e^{-i(2\pi/N)mk}\right\} \qquad (8)$$

where

$$f'(x_k) = f(x_k)e^{i(2\pi/N)kc}$$

and

$$c = (N - 1)/2 \quad .$$

The above equation defines the modified discrete Fourier transform, $F_a(u_m)$. According to this equation, to obtain a DFT which places the zero frequency in the center of the plot, one should premultiply the data samples $f(x_k)$ by a phase factor, perform the conventional DFT [see Eq. (2)], and then postmultiply the result by a second phase factor. In the following, the abbreviation MDFT will be used to designate the modified discrete Fourier transform defined in Eq. (8) above.

The two most important relations in connection with the DFT are: $U = 1/\Delta x$ and $\Delta u = 1/X$. U and Δu are adjusted simply by varying X and Δx. If $f(x)$ is identically zero outside $[-X/2, X/2]$, one can obtain finer resolution (Δu) in the frequency domain simply by extending the region of integration and filling the newly created space left and right of the old integration interval with zeros. It should also be carefully noted that if $f(x)$ is not identically zero outside $[-X/2, X/2]$, then greater accuracy in the calculation of $F(u_m)$ can be achieved by using more than just one term in approximating $f_p(x)$ in Eq. (5) [see Eqs. (5) and (6) and the intervening discussion].

By Eq. (1), the inverse Fourier transform is given by

$$f(x) = FT^{-1}\left\{F(u)\right\} = \int_{-\infty}^{\infty} F(u)e^{-i2\pi ux}du$$

$$\approx \int_{-U/2}^{U/2} F(u)e^{-i2\pi ux}du \quad ,$$

where U is chosen such that $F(u)$ is effectively zero outside the interval $[-U/2, U/2]$. Reasoning by analogy with Eq. (6), we obtain

$$f(x_k) \approx \int_{-U/2}^{U/2} F(u)e^{i2\pi ux_k}du$$

$$\approx \sum_{m=0}^{N-1} F(u_m)e^{i2\pi u_m x_k}\Delta u \equiv f_a(x_k) \quad .$$

Recalling that

$$u_m x_k = (1/N)(c^2 - mc - kc + mk) \qquad \text{[from before Eq. (8)]},$$

we obtain

$$f_a(x_k) = \sum_{m=0}^{N-1} F(u_m) e^{-i2\pi u_m x_k} \Delta u$$

$$= e^{i(2\pi/N)(c^2-kc)} \cdot (1/X) \cdot \left\{ \sum_{m=0}^{N-1} F'(u_m) e^{i(2\pi/N)km} \right\} \tag{9}$$

where

$$F'(u_m) = F(u_m) e^{-i(2\pi/N)mc}$$

The above equation defines the modified inverse discrete Fourier transform (MIDFT), $f_a(x_k)$. It should be compared with the conventional expression in Eq. (2).

From the theory of the (continuous) Fourier transform, it is known that $FT^{-1}\{FT[f(x)]\} \equiv f(x)$. We now show that an analogous relationship holds for the modified discrete Fourier transform; i.e., $MIDFT\{MDFT[f(x_k)]\} \equiv f(x_k)$:

$$MDFT[f(x_k)] = F_a(u_m) \qquad \text{[from Eq. (8)]}$$

$$g_a(x_k) \equiv MIDFT\{MDFT[f(x_k)]\}$$

$$= MIDFT\{F_a(u_m)\}$$

$$= e^{i(2\pi/N)(c^2-kc)} \cdot (1/X) \cdot \left\{ \sum_{m=0}^{N-1} [F_a(u_m) e^{-i(2\pi/N)mc}] e^{i(2\pi/N)km} \right\} \tag{10}$$

where the last line was obtained by using Eq. (9). Inserting the expression for $F_a(u_m)$ [Eq. (8)] into the braced expression above, we have

$$\{\ldots\} = \sum_{m=0}^{N-1} [F_a(u_m) e^{-i(2\pi/N)mc}] e^{i(2\pi/N)km}$$

$$= \sum_{m=0}^{N-1} [e^{-i(2\pi/N)(c^2-mc)} \cdot X \cdot \{(1/N) \sum_{k'=0}^{N-1} [f(x_{k'}) e^{i(2\pi/N)k'c}]$$

$$\cdot e^{-i(2\pi/N)mk'}\} e^{-i(2\pi/N)mc}]\ e^{i(2\pi/N)km}$$

$$= (X/N)e^{-i(2\pi/N)c^2} \sum_{k'=0}^{N-1} f(x_{k'})e^{i(2\pi/N)k'c} \underbrace{\left\{ \sum_{m=0}^{N-1} e^{i(2\pi/N)(c-k'-c+k)m} \right\}}_{N\delta_{kk'}}$$

$$= Xe^{-i(2\pi/N)c^2} f(x_k)e^{i(2\pi/N)kc} \quad .$$

Inserting this result into Eq. (10), we obtain

$$g_a(x_k) = \text{MIDFT}\left\{\text{MDFT}[f(x_k)]\right\}$$
$$= f(x_k) \quad .$$

Note the remarkable fact that the MDFT and the MIDFT, which are derived as approximations to the forward and inverse continuous Fourier transforms, turn out to be exact inverses of one another as well.

The convolution, h(x), of two functions f(x) and g(x) is given by

$$h(x) = f(x) * g(x)$$
$$= \int_{-\infty}^{\infty} f(x')g(x - x')dx'$$
$$= \int_{-\infty}^{\infty} f(x - x')g(x')dx'$$
$$H(u) = \int_{-\infty}^{\infty} h(x)e^{-i2\pi ux}dx$$
$$= \int_{-\infty}^{\infty} [\int_{-\infty}^{\infty} f(x - x')g(x')dx']e^{-i2\pi ux}dx \quad .$$

At this point, in pursuit of a numerical method for calculating the convolution integral using the MDFT, one might be tempted to replace the above integrals over x and x′ by discrete sums. If one chooses the sampling points $x_k = [k - (N - 1)/2]\Delta x$ and $x'_{k'} = [k' - (N - 1)/2]\Delta x$ as before, then the argument $x - x'$ (which is required in the convolution integral) goes over to $x_k - x'_{k'}$, which represents a set of points which interlace those defined by x_k and $x'_{k'}$. Because of this difficulty and also for reasons of simplicity, approximation of the above integrals by discrete sums is deferred until later. Interchanging the order of integration in the above expression for H(u), we obtain

$$H(u) = \int_{-\infty}^{\infty} dx'g(x') [\int_{-\infty}^{\infty} f(x - x')e^{-i2\pi ux}dx] \quad .$$

Calling the above bracketed expression I and setting $x'' = x - x'$, we obtain

$$I = \int_{-\infty}^{\infty} f(x'')e^{-i2\pi u(x' + x'')}dx''$$

$$= e^{-i2\pi ux'} \underbrace{\int_{-\infty}^{\infty} f(x'')e^{-i2\pi ux''}dx''}_{F(u)}$$

which gives $H(u) = F(u)\,G(u)$. The convolution integral $h(x) = f(x) * g(x)$ can therefore be computed by Fourier transforming f and g, forming the product of their transforms, and then inverse transforming the resulting product. To do the same calculation numerically, it would appear at first glance that one needs only to replace all continuous Fourier transforms (both forward and reverse) by their discrete counterparts and then to repeat the sequence of operations listed above. To understand clearly why the above prescription gives the wrong answer, consider the following. Assume that the functions f and g are both space-limited to the interval $[-X/2, X/2]$ (see Figure 2). Then

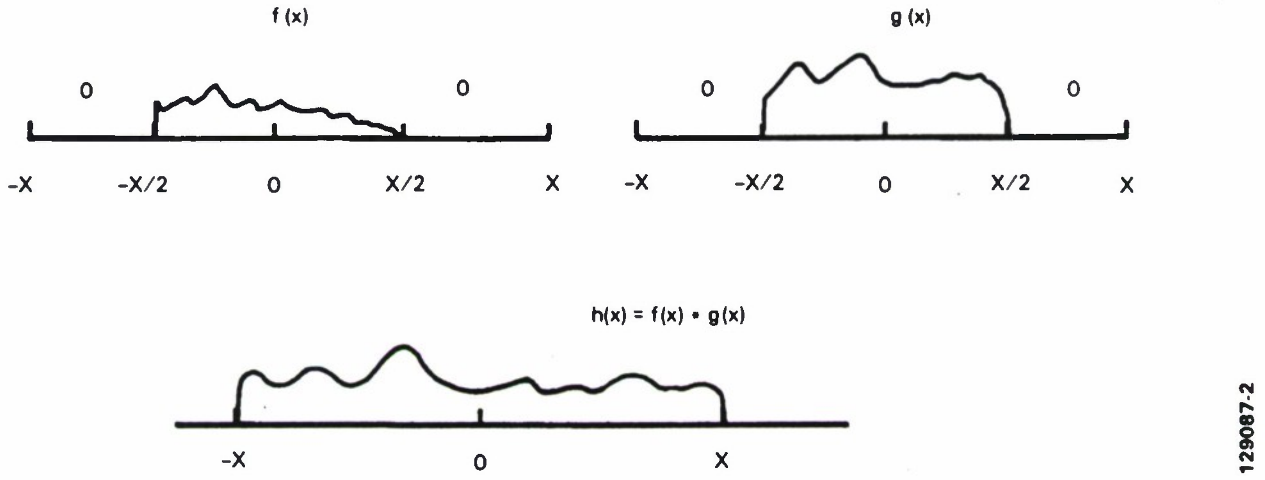


Figure 2. Relationship of the width of the convolution to that of its constituent functions.

their convolution $h = f * g$ is space-limited to $[-X, X]$, i.e., to an interval twice as long as that associated with either f or g. If one computes N-point modified discrete Fourier transforms of both f and g on the interval $[-X/2, X/2]$ and then calculates h by means of the procedure described immediately above, one obtains an h at N points on the interval $[-X/2, X/2]$ rather than at the 2N points required on the full interval $[-X, X]$. In addition, the values of h obtained at the N points are corrupted by what is commonly known as wraparound error.[10]

To obtain a good numerical approximation to $h(x) = f(x) * g(x)$ which is free of wraparound error on the full interval $[-X, X]$, one should proceed as follows:

a. Take both functions f and g to be space-limited to the interval $[-X/2, X/2]$ and evaluate each at the N equally spaced points $x_k = [k - (N-1)/2]X/N$, where $k = 0,1,...,N - 1$. Surround each sequence of N samples with N zeros, N/2 on the left and N/2 on the right (see Figure 2), so that f and g are now defined over the longer interval $[-X, X]$.

b. Compute the 2N-point MDFTs of the functions f and g defined above and then form the product of the two transforms.

c. Compute the 2N-point MIDFT of the product. The result is an approximation to the function $h(x) = f(x) * g(x)$ at the 2N equally spaced points $x_k = [k - (2N - 1)/2]X/N$, where $k = 0,1,...,2N - 1$. Since both truncation error and wraparound error have been eliminated from the above calculation, the former by the assumption that both f and g are space-limited and the latter by the symmetric padding of the original functions with zeros, the accuracy of the final results will be limited only by the rectangle-rule integration error associated with the computation of the required modified discrete Fourier transforms.

3. TWO-DIMENSIONAL FOURIER TRANSFORMS

The forward and inverse Fourier transforms in two dimensions are given, respectively, by

$$F(u, v) = FT\{f(x, y)\} = \iint_{-\infty}^{\infty} f(x, y)e^{-i2\pi(ux+vy)}dxdy$$

$$f(x,y) = FT^{-1}\{F(u,v)\} = \iint_{-\infty}^{\infty} F(u, v)e^{i2\pi(ux+vy)}dudv \quad .$$

By analogy with Section 2, we obtain the two-dimensional continuous Fourier transform in terms of the two-dimensional modified DFT by proceeding as follows:

Write the Fourier integral over the infinite plane as an infinite sum over rectangles of size XY. Choose the central rectangle (centered on the origin) large enough to ensure that the integral over this rectangle is already a good approximation to the integral over the infinite plane.

$$F(u, v) = \iint_{-\infty}^{\infty} f(x, y)e^{-i2\pi(ux+vy)}dxdy$$

$$= \sum_{m'=-\infty}^{\infty} \sum_{n'=-\infty}^{\infty} \int_{(m'-1/2)X}^{(m'+1/2)X} \int_{(n'-1/2)Y}^{(n'+1/2)Y} f(x, y)e^{-i2\pi(ux+vy)}dxdy \quad .$$

Now set $x' = x - m'X$ and $y' = y - n'Y$, interchange the order of summation and integration, and finally drop the primes on x' and y'. We then obtain

$$F(u, v) = \int_{-X/2}^{X/2} \int_{-Y/2}^{Y/2} \Big\{ \sum_{m'=-\infty}^{\infty} \sum_{n'=-\infty}^{\infty} f(x + m'X, y + n'Y)$$

$$\cdot\, e^{-i2\pi(um'X + vn'Y)}\Big\} e^{-i2\pi(ux + vy)}dxdy \quad .$$

By anology with the one-dimensional case, evaluate F(u,v) at the points

$$\begin{aligned} u_m &= [m - (N - 1)/2]/X \\ v_n &= [n - (N - 1)/2]/Y \end{aligned} \qquad m,n = 0, 1, \ldots, N - 1 \quad .$$

Substituting into the above equation, we obtain

$$F(u_m, v_n) = \int_{-X/2}^{X/2} \int_{-Y/2}^{Y/2} \underbrace{\Big\{ \sum_{m'=-\infty}^{\infty} \sum_{n'=-\infty}^{\infty} (-1)^{m'+n'} f(x+m'X, y+n'Y)\Big\}}_{\equiv f_p(x, y) \quad . \quad [\text{c.f. Eq.(5)}]} e^{-i2\pi(u_m x+v_n y)}dxdy$$

The above result is exact. By analogy with the one-dimensional case, choose the central rectangle large enough to ensure that the truncation error which results from setting $f_p(x, y) = f(x, y)$ is small:

$$F(u_m, v_n) = \int_{-X/2}^{X/2} \int_{-Y/2}^{Y/2} f_p(x, y)e^{-i2\pi(u_m x + v_n y)}dxdy$$

$$\approx \int_{-X/2}^{X/2} \int_{-Y/2}^{Y/2} f(x, y)e^{-i2\pi(u_m x + v_n y)}dxdy$$

$$\approx \sum_{k=0}^{N-1} \sum_{\ell=0}^{N-1} f(x_k, y_\ell)e^{-i2\pi(u_m x_k + v_n y_\ell)}\Delta x \Delta y \equiv F_a(u_m, v_n)$$

and the last line above follows from use of the midpoint rectangle rule in two dimensions, as illustrated in Figure 3. The two-dimensional forward transform is easily evaluated by iteration:

$$F_a(u_m, v_n) = \sum_{k=0}^{N-1} \sum_{\ell=0}^{N-1} f(x_k\ y_\ell)e^{-i2\pi(u_m x_k + v_n y_\ell)}\Delta x \Delta y$$

$$= \sum_{\ell=0}^{N-1} e^{-i2\pi v_n y_\ell}\Delta y \underbrace{\left[\sum_{k=0}^{N-1} f(x_k, y_\ell)e^{-i2\pi u_m x_k}\,\Delta x\right]}_{F_y(u_m,\ y_\ell)} . \tag{11}$$

Since the above quantity is simply a sequence of two one-dimensional transforms [see Eqs. (6) and (8)], one can write the discrete approximation without further calculation as:

$$F_a(u_m, v_n) \approx Ye^{-i(2\pi/N)(c^2 - nc)} \cdot \frac{1}{N} \sum_{\ell=0}^{N-1} \Big\{\Big\{ Xe^{-i(2\pi/N)(c^2 - mc)}$$

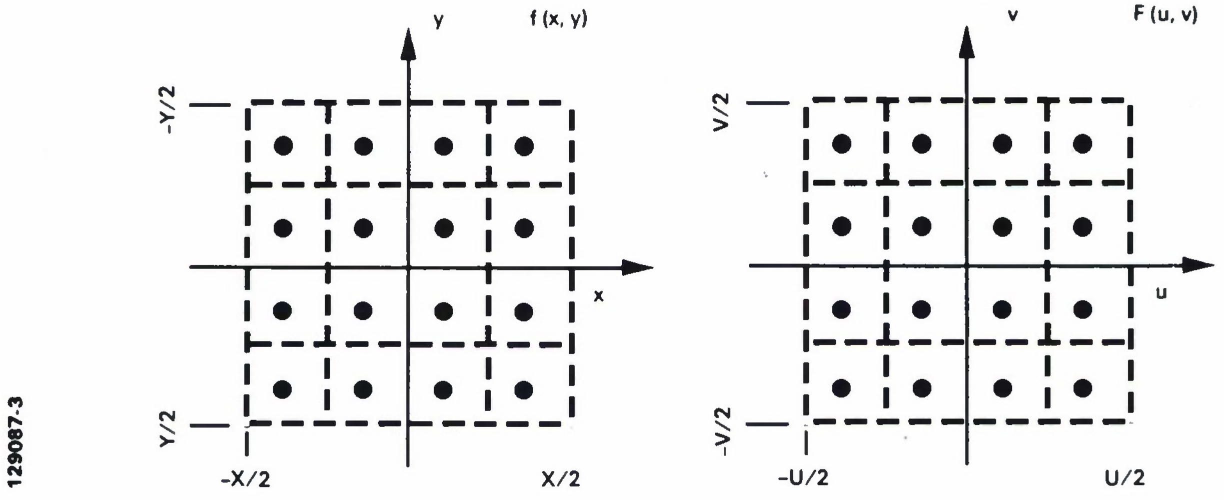
$$\cdot \frac{1}{N} \sum_{k=0}^{N-1} \left[f(x_k, y_\ell)e^{i(2\pi/N)kc}\right] \cdot e^{-i(2\pi/N)mk} \Big\{ e^{i(2\pi/N)\ell c} \Big\} e^{-i(2\pi/N)n\ell}$$

$$\approx XYe^{-i(2\pi/N)[2c^2 - (m+n)c]} \cdot \frac{1}{N^2} \sum_{k=0}^{N-1} \sum_{\ell=0}^{N-1} f'(x_k, y_\ell)\, e^{-i(2\pi/N)(mk + n\ell)} \tag{12}$$

where

$$f'(x_k, y_\ell) = f(x_k, y_\ell)e^{i(2\pi/N)(k+\ell)c} \quad .$$

The above equation defines the two-dimensional, modified discrete Fourier transform, $F_a(u_m, v_n)$.



$X = N\Delta x$	$U = N\Delta u$
$Y = N\Delta y$	$V = N\Delta v$
$\Delta x = 1/U$	$\Delta u = 1/X$
$\Delta y = 1/V$	$\Delta v = 1/Y$
$x_k = [k - (N - 1)/2]\Delta x$	$u_m = [m - (N - 1)/2]\Delta u$
$y_\ell = [1 - (N - 1)/2]\Delta y$	$v_n = [n - (N - 1)/2]\Delta v$

Figure 3. Calculation of the forward and inverse Fourier transforms in two dimensions.

Assume now that the function f(x, y) has been sampled on an N $\times$ N grid of points (N = even integer), as shown in Figures 3 and 4, and that the resulting data have been stored in a computer disk file on a row-by-row basis, with one row to a record. According to Eq. (11), a two-dimensional MDFT of the data can be obtained simply by subjecting each row to a one-dimensional Fourier transform and then repeating the procedure on the columns. In practice, the column transforms present a problem since the elements comprising a single column are stored in N separate records; retrieving from the disk the one piece of data required from each record turns out to be extremely time consuming. This problem is solved by using the Eklundh algorithm,[5] which makes possible the fast transposition of matrices of size N $\times$ N, provided that N is a power of two. How a fast matrix transposition solves this problem will become clear shortly.

With the above in mind, assume that the function f(x, y) has been sampled on an N $\times$ N grid of points, where N is now a power of two, and that the resulting data have been stored in a disk file after the manner described above. The required two-dimensional MDFT is then obtained as follows (see Figure 4):

a. Calculate the one-dimensional MDFT of each row of the matrix using the subroutine FFT1D. This routine evaluates Eq. (8) but omits the factor X. Note that FFT1D is contained within the subroutine FFT2D. After completion of the N transforms, the disk file contains the samples of $F_y(u_m, y_\ell)$.

b. Use the Eklundh fast matrix transposition algorithm, which is implemented in the subroutine TRANSP within FFT2D, to transpose the matrix. What were previously columns now become rows.

c. Reflect the elements of each row about the vertical axis. This ensures that the index ℓ increases from left to right (see Figure 4).

d. Repeat the use of FFT1D on each of the resulting rows.

e. Reflect, once again, the elements of each row about the vertical axis.

f. Transpose the matrix a second and final time. The disk file now contains the samples of $F(u_m, v_n)$. The data are stored on a row-by-row basis, with one row to a record.

The MIDFT in two dimensions is obtained from Eq. (9) in a manner completely analogous to that in which the forward transform in two dimensions was obtained from Eq. (8). We obtain

$$f_a(x_k, y_\ell) = (1/XY)e^{i(2\pi/N)[2c^2-(k+\ell)c]} \cdot \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} F'(u_m, v_n)e^{i(2\pi/N)(km+\ell n)} \tag{13}$$

where

$$F'(u_m, v_n) = F(u_m, v_n)e^{-i(2\pi/N)(m+n)c} \quad .$$

To calculate the MIDFT in two dimensions, in practice, one simply repeats the procedures a through f after altering the parameter list of FFT1D so that its output is the one-dimensional MIDFT.

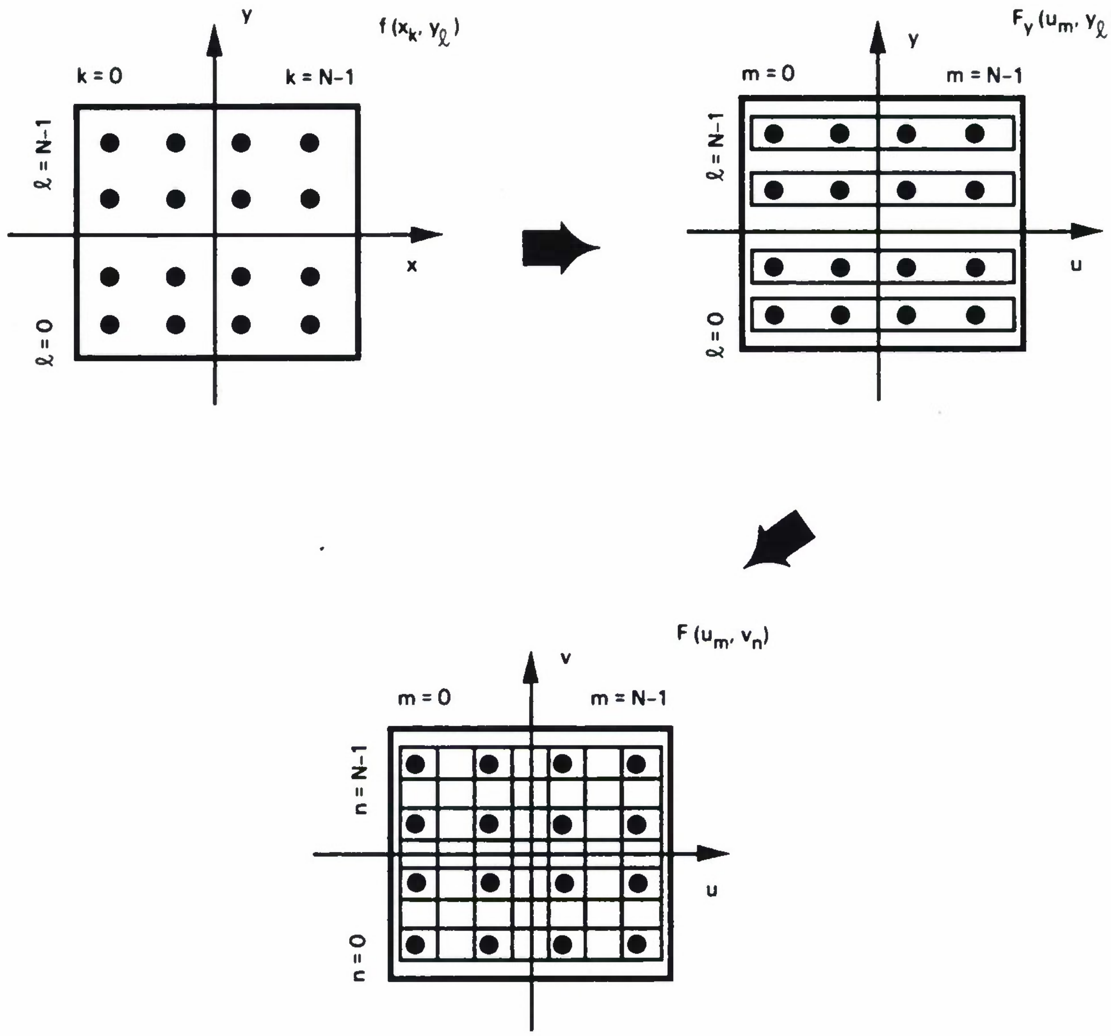


Figure 4. Calculation of the two-dimensional MDFT.

In two dimensions, as in one, the MDFT and MIDFT are exact inverses of one another. The proof is straightforward.

Two-dimensional convolution is completely analogous to that in one dimension [see the program CONVOLVE listed in the Appendix and described in Section 4.4]. To prevent wraparound error in the computation of $h(x, y) = f(x, y) * g(x, y)$, one proceeds as follows. Assume that one has two matrices of size $N \times N$ which contain samples of f and g. Each matrix is expanded to size $2N \times 2N$ by placing the initial array into the center of a $2N \times 2N$ array and then filling the "collar" region with zeros. The two-dimensional convolution, free of wraparound error, is then obtained by Fourier transforming the two new $2N \times 2N$ matrices, multiplying the two transforms on a point-by-point basis, and finally inverse transforming the resulting matrix.

4. PROGRAM DESCRIPTIONS AND SAMPLE CALCULATIONS

The Appendix contains printouts of four FORTRAN programs described below: FFT2D, TTRANSP, TFFT2D, and CONVOLVE.

4.1 FFT2D

The subroutine FFT2D calculates either the forward or inverse, modified (in the sense discussed above), two-dimensional, discrete Fourier transform using the fast Fourier transform algorithm. The subroutine accepts as input the single precision, real data stored in two separate disk files, say REAL.DAT and IMAG.DAT, and assumes that these data represent the real and imaginary parts, respectively, of the samples of the function to be transformed. Each file contains N records of N samples each, with each record containing one row of the matrix of samples. N is limited to a power of two ($N = 2^{NPOW}$) and must, in addition, satisfy the inequality $4 \leqslant N \leqslant 2^{20}$.

If in the parameter list of FFT2D we set IDIR = +1, then a Fourier transform in the forward direction is computed. The subroutine assumes, in this case, that the input data contained in REAL.DAT and IMAG.DAT represent the real and imaginary parts, respectively, of the samples of the complex function f(x, y) on the N $\times$ N grid of points

$$(x_k, y_\ell) = [(k - \frac{N-1}{2})\frac{X}{N}, (\ell - \frac{N-1}{2})\frac{Y}{N}] \quad , \tag{14}$$

where $k,\ell = 0, 1,\ldots,N-1$ and X and Y are as in Figure 3. The sampling indices (k,ℓ) are related to the row and column indices (ROW, COL) of the input files via

k = COL - 1
ℓ = N - ROW

If the input function f(x, y) is real, then the file IMAG.DAT must be filled with zeros. Using the calculation procedure outlined in Section 3(a) through (f), FFT2D returns, in files REAL.DAT and IMAG.DAT, the real and imaginary parts, respectively, of the samples of the Fourier transform function F(u,v) on the N $\times$ N grid of points

$$(u_m, v_n) = [(m - \frac{N-1}{2})\frac{U}{N}, (n - \frac{N-1}{2})\frac{V}{N}] \quad , \tag{15}$$

where m, n = 0, 1,...,N - 1, and U and V are as in Figure 3. Note, according to this figure, that $U/N = \Delta u = 1/X$ and $V/N = \Delta v = 1/Y$. The sampling indices (m, n) are easily seen to be related to the row and column indices (ROW, COL) of the output files via

m = COL - 1
n = N - ROW .

Mathematically, we have

$$F(u_m, v_n) = e^{-i(2\pi/N)[2c^2 - (m + n)c]}$$

$$\cdot \frac{1}{N^2} \sum_{k=0}^{N-1} \sum_{\ell=0}^{N-1} \left[f(x_k, y_\ell) e^{i(2\pi/N)(k + \ell)c} \right] e^{-i(2\pi/N)(mk + n\ell)} \quad . \tag{16}$$

Note carefully that the above quantity calculated by FFT2D differs from the two-dimensional MDFT of Eq. (12) by the omission of the leading factor XY.

If in the parameter list of FFT2D we set IDIR = –1, then an *inverse* Fourier transform is computed. The subroutine assumes, in this case, that the input data contained in REAL.DAT and IMAG.DAT represent the real and imaginary parts, respectively, of the samples of the complex function F(u,v) on the N × N grid of points (u_m, v_n), as defined previously. On output, the files contain the real and imaginary parts of the samples of the inverse Fourier transform function f(x,y) on the N × N grid of points (x_k, y_ℓ), also as defined previously. Mathematically, we have in this inverse case

$$f(x_k, y_\ell) = e^{i(2\pi/N)[2c^2 - (k + \ell)c]}$$

$$\cdot \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} \left[F(u_m, v_n) e^{-i(2\pi/N)(m + n)c} \right] \cdot e^{i(2\pi/N)(km + \ell n)} \quad . \tag{17}$$

Note here that the above inverse quantity calculated by FFT2D differs from the two-dimensional MIDFT of Eq. (13) by the omission of the leading factor 1/XY.

According to Section 3(a), both the forward and inverse, modified, one-dimensional DFTs are computed by the subroutine FFT1D, which implements Eqs. (8) and (9) (minus their factor X). This routine, in turn, calls directly or indirectly three additional routines named FASTF, FASTG, and SCRAM. These three routines, which calculate the forward and inverse conventional, one-dimensional DFTs, were written by Donald Monro[11] of Imperial College and are republished here by permission of the Royal Statistical Society, London.

A quick comparison of the array storage requirements for FFT2D and TRANSP shows that nearly all the array space required by FFT2D is required by the component subroutine TRANSP, which performs fast matrix transposition using the Eklundh algorithm. In fact, of the approximately 4N(M + MPOW + 10) bytes of array storage required by FFT2D, a full 4N(M + MPOW + 1) bytes are needed to execute TRANSP. In these expressions, N is the dimension of the matrix, which must be a power of two: $N = 2^{NPOW}$. In addition, M is the number of rows of the matrix operated on in the internal memory of the computer at one time by TRANSP; it must also be a power of two: $M = 2^{MPOW}$. Note that TRANSP will execute properly (though slowly) with M as small as 2, or equivalently, MPOW = 1. The ratio R = (NPOW/MPOW) is very important in practice since it determines the amount of I/O required by subroutine TRANSP within FFT2D. As a glance at the first few lines of this component subroutine will make clear, the number of I/O passes (where an I/O pass is defined as reading in and writing out all the data exactly once) required by TRANSP is equal to [R], where [R] is shorthand for the smallest integer equal to, or larger than, R. Thus, by way of example, [4/3] = 2 and [10/5] = 2. In practice, [R] is bounded below by 2 since [R] = [NPOW/MPOW] = 1 would imply that NPOW = MPOW or equivalently M = N. In such a case, the subroutine TRANSP would be superfluous. Accordingly, one expects no reduction in TRANSP I/O time for values of MPOW exceeding [NPOW/2]. If, in addition, the computation time required by TRANSP is only weakly dependent on MPOW, one should expect no reduction in total execution time for TRANSP for MPOW exceeding [NPOW/2]. This expectation is fully confirmed by timing tests. More importantly, what is true for TRANSP turns out to be true for FFT2D as a whole. For a matrix of size N $\times$ N, where N = 2^{NPOW}, one finds no reduction in total execution time for FFT2D for MPOW exceeding [NPOW/2].

To make the foregoing points more concrete, consider an example. For a complex matrix of size 1024 $\times$ 1024 (represented in actuality by two 1024 $\times$ 1024 real arrays contained in the disk files REAL.DAT and IMAG.DAT), we have N = 1024 = 2^{10} and, therefore, NPOW = 10. According to the above considerations, to minimize execution time without wasting any array storage space, one should choose MPOW = 5. The internal memory required by FFT2D is then given by $4N(2^{MPOW} + MPOW + 10) \approx 190$ KB. To this, of course, must be added the space required by the subroutine package FFT2D and the calling program. Choosing MPOW smaller than 5 will reduce the internal memory required but also increase the execution time. On the basis of the above, it is clear that with FFT2D, a 1024 $\times$ 1024 complex matrix can easily be Fourier transformed on a microcomputer having a modest internal memory and a hard disk of no more than ~10-MB capacity. It should be noted carefully, however, that because of the very heavy I/O demands made by FFT2D, it is extremely important to have a disk with the smallest possible mean access time. This conclusion follows upon noting that the time required to transfer, say, 4N = 4096 bytes of data between computer and disk at a rate of 600 KB/s (typical of microcomputer hard disks) is ~7 ms; this time, in turn, is much smaller than the mean access time characteristic of most microcomputer hard disks.

4.2 TTRANSP

TTRANSP is a test program which allows a prospective user of FFT2D to verify that the fast matrix transposition subroutine, TRANSP (the last subroutine included in the FFT2D package) functions without error. For each case (NPOW, MPOW), where NPOW = 2, ..., 10 and MPOW = 1, ..., NPOW, the program generates a matrix of size N $\times$ N (= $2^{NPOW} \times 2^{NPOW}$) containing consecutive integers from 1 to N^2, transposes the matrix by means of the subroutine TRANSP, and then compares the elements of the transposed matrix on a point-by-point basis with the quantities which should be there if the transposition has been achieved without error. TTRANSP was run on our DEC MicroVAX II computer with the result that TRANSP was shown to function flawlessly in all 54 cases tested. This includes matrices up to 1024 $\times$ 1024 in size.

4.3 TFFT2D

TFFT2D allows the user to test the accuracy of FFT2D in performing both forward and inverse transforms. The forward transform capability is tested by means of a three-step procedure. First, function subprograms FR and FI, representing the real and imaginary parts of the complex function f(x, y) to be transformed, are used to generate samples of f(x, y) on the grid of Eq. (14) with N = 256 and the samples so produced are then stored in two separate disk files, REAL.DAT and IMAG.DAT. Second, the disk files are input to FFT2D and a 256×256 forward MDFT of the input data is computed. Upon completion of the execution of FFT2D, the files REAL.DAT and IMAG.DAT contain the real and imaginary parts, respectively, of the transformed function F(u, v) evaluated on the grid of points given in Eq. (15). Third, at a small number of points (u_m, v_n), or equivalently (ROW, COL), specified at the beginning of the program, the Fourier transformed function F(u, v) is computed directly, but slowly, using Eq. (16); these quantities are then compared on a point-by-point basis with the corresponding quantities calculated by FFT2D. Although FFT2D performs all calculations in single precision, the direct, slow calculation must be done in double precision because the roundoff error in this case propagates much more rapidly than it does in the case of FFT2D.

As listed in the Appendix, TFFT2D produces the following output in connection with its test of the forward Fourier transform:

```
ROW COL   RE(SLOW FT)   RE(FFT)    REL. ERR.    IM(SLOW FT)    IM(FFT)    REL. ERR.
--- ---   -----------   -------    ---------    -----------    -------    ---------


128 151   -0.8556D-03 -0.8556E-03 -0.14D-06    -0.2546D-10  0.2024E-09 -0.89D+01
128 152   -0.8529D-03 -0.8529E-03 -0.14D-06    -0.8735D-10  0.3375E-09 -0.49D+01
128 153   -0.8191D-03 -0.8191E-03 -0.28D-06    -0.1613D-09  0.1217E-09 -0.18D+01
128 154   -0.7577D-03 -0.7577E-03 -0.77D-07     0.1066D-09  0.2260E-09  0.11D+01
```

In our experience, the numerical results above, with relative errors of approximately 10^{-7}, are typical of what one encounters in practice with arrays up to 1024×1024 in size. It should be borne in mind, however, that relative errors can be much larger at a sampling point (u_m, v_n) where the Fourier transform F has a very steep slope. In our tests, we have observed relative errors as large as approximately 10^{-5} at such points. Note that the relative errors computed in the case of the imaginary quantities above are irrelevant since F(u, v) is a real quantity.

Since the function f(x, y), which is subjected to discrete Fourier transformation in TFFT2D via both FFT2D and direct calculation, can be transformed analytically as well, we compare the three results at one point, say ROW = 128 and COL = 151. The function represented by the two function subprograms FR and FI [where f(x, y) = FR(x, y) + iFI(x,y)] is obviously

$$f(x, y) = \mathrm{rect}(\frac{16x}{X})\ \mathrm{rect}(\frac{16y}{Y}) \quad ,$$

where rect(x) = 1 for $-1/2 < x < 1/2$, and 0 otherwise. The Fourier transform is easily calculated to be

$$F(u, v) = \frac{XY}{256}\ \mathrm{sinc}(\frac{uX}{16})\ \mathrm{sinc}(\frac{vY}{16}) \quad ,$$

where sinc(x) = $(\sin \pi x)/\pi x$. To obtain F(u, v) at the array point (ROW, COL) = (128, 151), we must first determine u_m and v_n at this point. From Eq. (15a) and Figure 3, we have

$$m = \text{COL} - 1$$

$$n = N - \text{ROW}$$

$$u_m = [m - \frac{N-1}{2}]\frac{1}{X}$$

$$v_n = [n - \frac{N-1}{2}]\frac{1}{Y} \quad .$$

Setting N = 256, ROW = 128, and COL = 151, we obtain

$$m = 150$$

$$n = 128$$

$$u_{150}X = (150 - \frac{255}{2}) = 22.5$$

$$v_{128}Y = (128 - \frac{255}{2}) = 0.5 \quad .$$

Hence, the output array cell (ROW, COL) = (128, 151) contains

$$F(u_{150}, v_{128}) = \frac{XY}{256} \text{ sinc}(\frac{u_{150}X}{16}) \text{ sinc}(\frac{v_{128}Y}{16}) = (-0.8448 \times 10^{-3})XY \quad .$$

Thus, the analytic result and numerically calculated results differ by approximately one percent in this case. The functions f(x, y) and F(u, v) (the latter calculated by FFT2D) are graphed in Figures 5(a) and (b).

In the latter part of the program TFFT2D, the inverse transform capability of FFT2D is tested by invoking FFT2D with IDIR = –1 to inverse Fourier transform the data left in the disk files REAL.DAT and IMAG.DAT as a result of earlier invocation of FFT2D with IDIR = + 1 (forward transform). Since these two discrete transforms are *exact* inverses of one another, we expect the inverse transform to return us to the sampled version of f(x, y) with which we started. The following output is produced by this final part of the program:

```
MIN ABS ERROR OF REAL PART =   0.0000E+00

MAX ABS ERROR OF REAL PART =   0.4768E-06

MEAN ABS ERROR OF REAL PART =   0.6134E-08

MIN ABS ERROR OF IMAGINARY PART =   0.0000E+00

MAX ABS ERROR OF IMAGINARY PART =   0.3504E-06

MEAN ABS ERROR OF IMAGINARY PART =   0.5537E-08

^Z
```

where sinc(x) = $(\sin \pi x)/\pi x$. To obtain F(u, v) at the array point (ROW, COL) = (128, 151), we must first determine u_m and v_n at this point. From Eq. (15a) and Figure 3, we have

$$m = COL - 1$$

$$n = N - ROW$$

$$u_m = [m - \frac{N-1}{2}]\frac{1}{X}$$

$$v_n = [n - \frac{N-1}{2}]\frac{1}{Y} \quad .$$

Setting N = 256, ROW = 128, and COL = 151, we obtain

$$m = 150$$

$$n = 128$$

$$u_{150}X = (150 - \frac{255}{2}) = 22.5$$

$$v_{128}Y = (128 - \frac{255}{2}) = 0.5 \quad .$$

Hence, the output array cell (ROW, COL) = (128, 151) contains

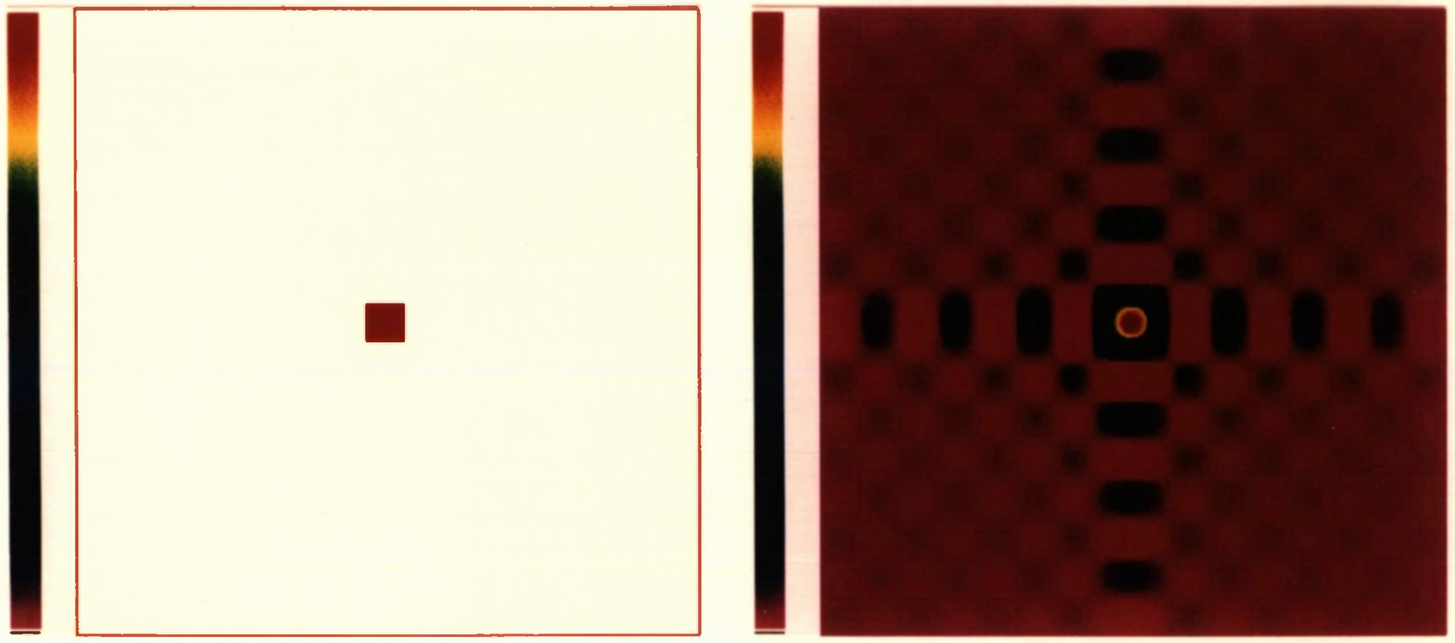
$$F(u_{150},v_{128}) = \frac{XY}{256} \text{ sinc}(\frac{u_{150}X}{16}) \text{ sinc}(\frac{v_{128}Y}{16}) = (-0.8448 \times 10^{-3})XY \quad .$$

Thus, the analytic result and numerically calculated results differ by approximately one percent in this case. The functions f(x, y) and F(u, v) (the latter calculated by FFT2D) are graphed in Figures 5(a) and (b).

In the latter part of the program TFFT2D, the inverse transform capability of FFT2D is tested by invoking FFT2D with IDIR = -1 to inverse Fourier transform the data left in the disk files REAL.DAT and IMAG.DAT as a result of earlier invocation of FFT2D with IDIR = + 1 (forward transform). Since these two discrete transforms are *exact* inverses of one another, we expect the inverse transform to return us to the sampled version of f(x, y) with which we started. The following output is produced by this final part of the program:

```
MIN ABS ERROR OF REAL PART =    0.0000E+00

MAX ABS ERROR OF REAL PART =    0.4768E-06

MEAN ABS ERROR OF REAL PART =    0.6134E-08

MIN ABS ERROR OF IMAGINARY PART =    0.0000E+00

MAX ABS ERROR OF IMAGINARY PART =    0.3504E-06

MEAN ABS ERROR OF IMAGINARY PART =    0.5537E-08

^Z
```

129087-5



(a)

Function $f(x, y) = \text{rect}\left(\frac{16x}{X}\right) \text{rect}\left(\frac{16y}{Y}\right)$ as evaluated by TFFT2D on a 256×256 grid

(b)

The function $F(u, v) = \text{MDFT}\ [f(x, y)]$ as calculated by TFFT2D

Figure 5. A function and its Fourier transform.

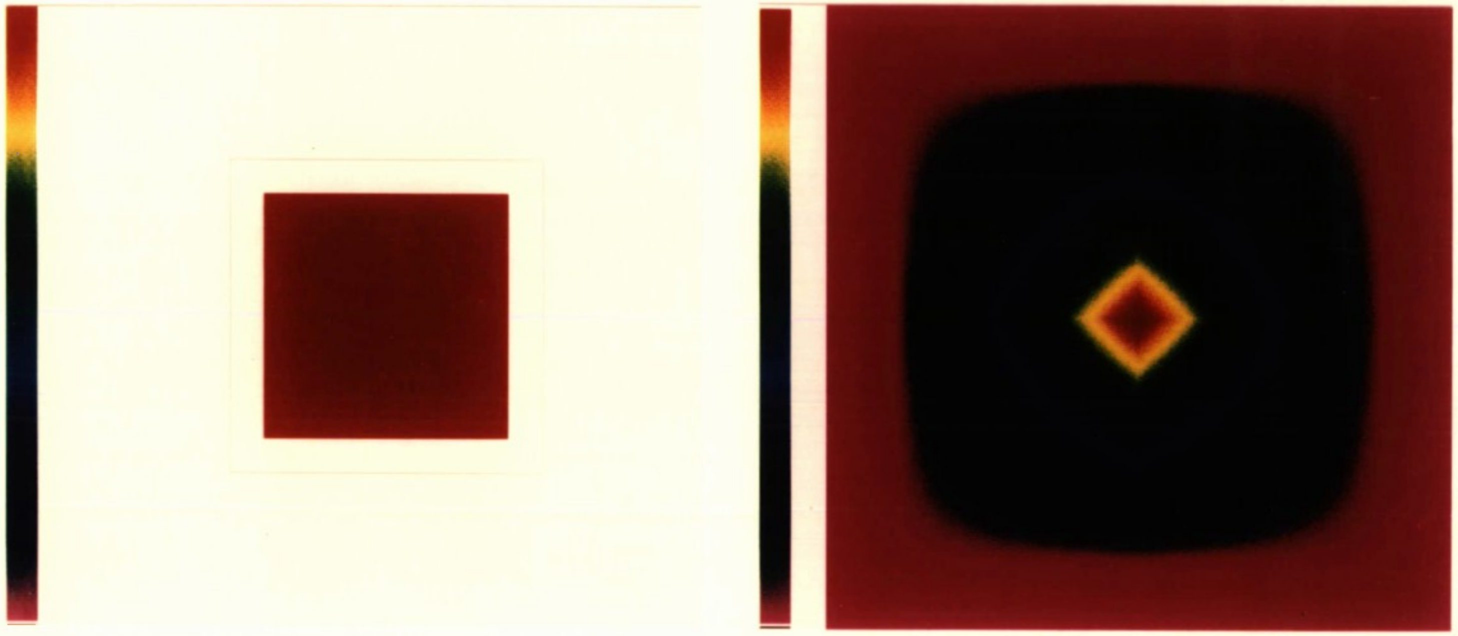
Before leaving the subject of testing the subroutine FFT2D, we note that it has been run extensively on the DEC MicroVAX II computer in our laboratory. Mass storage for the computer is provided by RD-53 and RD-54 disk memory units, both of which have mean access times of 38 ms and transfer rates of 625 KB/s. Below we list typical CPU and total execution times for FFT2D on this computer as a function of complex array size and MPOW. Note that the total execution time includes I/O time whereas the CPU time does not.

Size		CPU Time	Total Execution
(N × N)	(NPOW, MPOW)	(s)	Time (s)
256 × 256	(8, 4)	82	264
512 × 512	(9, 5)	320	696
1024 × 1024	(10, 5)	1347	2222

4.4 CONVOLVE

This program computes the convolution of two complex 512 × 512 arrays using procedures 3(a) through f. For example, we choose f(x, y) = rect[(512/400)(x/X)] rect[(512/400)(y/Y)]. When sampled on a 512 × 512 grid, this function appears as a 400 × 400 array of ones in the center of a field of zeros, as ilustrated in Figure 6(a). Figure 6(b), in turn, shows the 1024 × 1024 array which results from using CONVOLVE to convolve the array of Figure 6(a) with itself.

129087-6



Function $f(x, y) = \mathrm{rect}\left(\frac{512}{400}\frac{x}{X}\right)\,\mathrm{rect}\left(\frac{512}{400}\frac{y}{Y}\right)$ as evaluated on a 512×512 grid

The 1024×1024 array which results from using CONVOLVE to convolve (a) with itself

Figure 6. A function and its auto-convolution.

5. CONCLUSION

A simplified treatment of the one-dimensional DFT has been provided and applied to the case of the two-dimensional DFT. Specific rules for computer implementation using fast algorithms for Fourier transformation and matrix transposition have also been given. Note that the formulation given here for the discrete transforms differs from general convention in that the basic discrete interval is placed symmetrically about zero in both direct and transformed spaces.

REFERENCES

1. E.O. Brigham, *The Fast Fourier Transform*, Englewood Cliffs, N.J.: Prentice-Hall (1974), Ch. 6.
2. P.J. Davis and P. Rabinowitz, *Methods of Numerical Integration*, New York: Academic Press (1975), pp.185-201.
3. J.W. Cooley, P.A.W. Lewis, and P.D. Welch, "Application of the fast Fourier transform to computation of Fourier integrals, Fourier series, and convolution integrals," *IEEE Trans. Audio Electroacous.* **AU-15,** 79-84 (June 1967).
4. Brigham, *op. cit.,* p. 134.
5. J.O. Eklundh, "A fast computer method for matrix transposing," *IEEE Trans. Comput.* **C-21,** 801-3 (July 1972).
6. M.H. Serzu and W.M. Moon, "Two-dimensional fast Fourier transform for large data matrices," *Computer Phys. Commun.* **52,** 333-6 (1989).
7. G.L. Anderson, "A stepwise approach to computing the multidimensional fast Fourier transform of large arrays," *IEEE Trans. Acoust. Speech Signal Process.* **ASSP-28,** 280-4 (1980).
8. R.J. Collier, C.B. Burckhardt, and L.H. Lin, *Optical Holography*, New York: Academic Press (1971), Ch. 19.
9. Brigham, *op. cit.,* Sec. 5-3.
10. Brigham, *op. cit.,* Sec. 7-3.
11. D.M. Monro, "Complex discrete fast Fourier transform," *Appl. Statist.* **24,** 153-60 (1975).

APPENDIX

```
      SUBROUTINE FFT2D(IDEV1, IMARK1, IDEV2, IMARK2, IDIR, MPOW, M,
     & NPOW, N, NHALF, IP, IPP, IPAIR, IROWNR, ROWS, XREAL, XIMAG,
     & DC, DS, DCC, DSS)
C
C THIS SUBROUTINE CALCULATES EITHER THE FORWARD OR INVERSE, MODIFIED,
C TWO-DIMENSIONAL DFT USING THE FFT ALGORITHM AND THE EKLUNDH FAST
C MATRIX TRANSPOSITION ALGORITHM.
C
      INTEGER IP(MPOW, NHALF), IPP(MPOW, NHALF), IPAIR(MPOW),
     & IROWNR(M)
      REAL ROWS(M, N), XREAL(N), XIMAG(N)
      DOUBLE PRECISION DC(N), DS(N), DCC(N), DSS(N), PI,
     & DFAC1, DFAC2, DARG1, DARG2
C
      DATA PI /3.1415926535897932D0/
      DFAC1 = (2.0D0 * PI) / DFLOAT(N)
      DFAC2 = DFLOAT(N - 1) / 2.0D0
C
C CALCULATE PRE- AND POST-MULTIPLICATION FACTORS USED IN CALCULATING
C MODIFIED, ONE-DIMENSIONAL DFT
C
      DO 10 I = 1, N
         DARG1 = DFAC1 * DFAC2 * DFLOAT(I - 1)
         DARG2 = DARG1 - DFAC1 * DFAC2 * DFAC2
         DC(I) = DCOS(DARG1)
         DS(I) = DSIN(DARG1)
         DCC(I) = DCOS(DARG2)
         DSS(I) = DSIN(DARG2)
10    CONTINUE
C
C CALCULATE MODIFIED, ONE-DIMENSIONAL DFTS OF INDIVIDUAL ROWS
C
      DO 20 I = 1, N
         READ (IDEV1, REC = I) XREAL
         READ (IDEV2, REC = I) XIMAG
         CALL FFT1D(XREAL, XIMAG, DC, DS, DCC, DSS, N, IDIR)
         WRITE (IDEV1, REC = I) XREAL
         WRITE (IDEV2, REC = I) XIMAG
20    CONTINUE
C
C TRANSPOSE COMPLEX MATRIX (STORED IN TWO SEPARATE FILES)
C
      CALL TRANSP(IDEV1, IMARK1, MPOW, M, NPOW, N, NHALF, IP, IPP,
     & IPAIR, IROWNR, ROWS, XREAL)
      CALL TRANSP(IDEV2, IMARK2, MPOW, M, NPOW, N, NHALF, IP, IPP,
     & IPAIR, IROWNR, ROWS, XIMAG)
C
C "REFLECT" EACH ROW ABOUT CENTER AND CALCULATE MODIFIED, ONE-DIMENSIONAL
C TRANSFORM; "REFLECT" SECOND TIME ABOUT CENTER
C
      DO 50 I = 1, N
         READ (IDEV1, REC = I) XREAL
         READ (IDEV2, REC = I) XIMAG
         DO 30 J = 1, NHALF
            TEMPREAL = XREAL(J)
            TEMPIMAG = XIMAG(J)
            XREAL(J) = XREAL(N + 1 - J)
            XIMAG(J) = XIMAG(N + 1 - J)
            XREAL(N + 1 - J) = TEMPREAL
            XIMAG(N + 1 - J) = TEMPIMAG
```

```
30         CONTINUE
           CALL FFT1D(XREAL, XIMAG, DC, DS, DCC, DSS, N, IDIR)
           DO 40 J = 1, NHALF
              TEMPREAL = XREAL(J)
              TEMPIMAG = XIMAG(J)
              XREAL(J) = XREAL(N + 1 - J)
              XIMAG(J) = XIMAG(N + 1 - J)
              XREAL(N + 1 - J) = TEMPREAL
              XIMAG(N + 1 - J) = TEMPIMAG
40         CONTINUE
           WRITE (IDEV1, REC = I) XREAL
           WRITE (IDEV2, REC = I) XIMAG
50      CONTINUE
C
C TRANSPOSE COMPLEX MATRIX SECOND AND FINAL TIME
C
        CALL TRANSP(IDEV1, IMARK1, MPOW, M, NPOW, N, NHALF, IP, IPP,
     &  IPAIR, IROWNR, ROWS, XREAL)
        CALL TRANSP(IDEV2, IMARK2, MPOW, M, NPOW, N, NHALF, IP, IPP,
     &  IPAIR, IROWNR, ROWS, XIMAG)
C
        RETURN
        END
C
C
C ***********************************************************************
C
C
        SUBROUTINE FFT1D(XREAL, XIMAG, DC, DS, DCC, DSS, N, IDIR)
C
C THIS SUBROUTINE CALCULATES EITHER THE FORWARD OR INVERSE, MODIFIED,
C ONE-DIMENSIONAL DFT USING THE FFT ALGORITHM.
C
        REAL XREAL(N), XIMAG(N)
        DOUBLE PRECISION DC(N), DS(N), DCC(N), DSS(N), DXREAL,
     &  DXIMAG, DIDIR
C
        DIDIR = DFLOAT(IDIR)
C
C MULTIPLY INPUT DATA BY PRE-MULTIPLICATION PHASE FACTOR
C
        DO 10 I = 1, N
           DXREAL = XREAL(I)
           DXIMAG = XIMAG(I)
           XREAL(I) = DXREAL * DC(I) - DIDIR * DXIMAG * DS(I)
           XIMAG(I) = DXIMAG * DC(I) + DIDIR * DXREAL * DS(I)
10      CONTINUE
C
C CALCULATE CONVENTIONAL, ONE-DIMENSIONAL DFT
C
        CALL FASTF(XREAL, XIMAG, N, IDIR)
C
C MULTIPLY TRANSFORMED DATA BY POST-MULTIPLICATION PHASE FACTOR
C
        DO 20 I = 1, N
           DXREAL = XREAL(I)
           DXIMAG = XIMAG(I)
           XREAL(I) = DXREAL * DCC(I) - DIDIR * DXIMAG * DSS(I)
           XIMAG(I) = DXIMAG * DCC(I) + DIDIR * DXREAL * DSS(I)
20      CONTINUE
```

```
C
        RETURN
        END
C
C
C ***********************************************************************
C
C
        SUBROUTINE FASTF(XREAL, XIMAG, ISIZE, ITYPE)
C
C SOURCE:  DONALD M. MONRO, "COMPLEX DISCRETE FAST FOURIER
C          TRANSFORM", APPLIED STATISTICS 24(1975)153-160.
C
C THIS SUBROUTINE CALCULATES EITHER THE FORWARD OR INVERSE, CONVENTIONAL,
C ONE-DIMENSIONAL DFT OF A SEQUENCE OF LENGTH ISIZE -- WHERE ISIZE IS A
C POWER OF 2 -- USING THE SANDE-TUKEY VERSION OF THE FFT ALGORITHM.
C
C-----------------------------------------------------------------------
C
C PARAMETERS:
C
C XREAL(ISIZE)   INPUT:  THE REAL PART OF THE ORIGINAL SEQUENCE.
C                OUTPUT:  THE REAL PART OF THE TRANSFORM.
C
C XIMAG(ISIZE)   INPUT:  THE IMAGINARY PART OF THE ORIGINAL SEQUENCE.
C                OUTPUT:  THE IMAGINARY PART OF THE TRANSFORM.
C
C ISIZE          INTEGER LENGTH OF TRANSFORM.  MUST BE POSITIVE
C                AND A POWER OF 2.  MUST ALSO BE GREATER THAN OR EQUAL
C                TO 4 AND LESS THAN OR EQUAL TO 2 ** 20.
C
C ITYPE          TYPE OF TRANSFORM AS INDICATED BY AN INTEGER
C                CONSTANT.  A FORWARD TRANSFORM IS FOUND IF
C                ITYPE IS POSITIVE, INVERSE IF NEGATIVE.  IF
C                ITYPE IS ZERO, NO TRANSFORM IS DONE.
C
C-----------------------------------------------------------------------
C
C RADIX 4 COMPLEX DISCRETE FAST FOURIER TRANSFORM WITH
C UNSCRAMBLING OF THE TRANSFORMED ARRAYS
C
        REAL XREAL(ISIZE), XIMAG(ISIZE)
C
C CHECK FOR VALID TRANSFORM SIZE
C
        II = 4
C
        DO 20 K = 2, 20
           IPOW = K
           IF (II - ISIZE) 10, 40, 30
10         II = II * 2
20      CONTINUE
C
C IF THIS POINT IS REACHED A SIZE ERROR HAS OCCURRED
C
30      RETURN
C
C CALL FASTG TO PERFORM THE TRANSFORM
C
40      CALL FASTG(XREAL, XIMAG, ISIZE, ITYPE)
```

```
C
C CALL SCRAM TO UNSCRAMBLE THE RESULTS
C
        CALL SCRAM(XREAL, XIMAG, ISIZE, IPOW)
C
        RETURN
        END
C
C
C ***********************************************************************
C
C
        SUBROUTINE FASTG(XREAL, XIMAG, N, ITYPE)
C
C SOURCE:  DONALD M. MONRO, "COMPLEX DISCRETE FAST FOURIER
C          TRANSFORM", APPLIED STATISTICS 24(1975)153-160.
C
C RADIX 4 COMPLEX DISCRETE FAST FOURIER TRANSFORM WITHOUT UNSCRAMBLING,
C SUITABLE FOR CONVOLUTIONS OR OTHER APPLICATIONS WHICH DO NOT REQUIRE
C UNSCRAMBLING.  SUBROUTINE FASTF USES THIS ROUTINE FOR TRANSFORMATION
C AND ALSO PROVIDES UNSCRAMBLING.
C
        REAL XREAL(N), XIMAG(N)
        DATA ZERO, HALF, ONE, ONE5, TWO, FOUR /0.0, 0.5, 1.0, 1.5,
     &       2.0, 4.0/
        DATA PIE /3.141592654/
        IFACA = N / 4
        IF (ITYPE) 10, 170, 30
C
C IF THIS IS TO BE AN INVERSE TRANSFORM, CONJUGATE THE DATA
C
10      DO 20 K = 1, N
           XIMAG(K) = -XIMAG(K)
20      CONTINUE
C
30      IFCAB = IFACA * 4
C
C DO THE TRANSFORMS REQUIRED BY THIS STAGE
C
        Z = PIE / FLOAT(IFCAB)
        BCOS = -TWO * SIN(Z) ** 2
        BSIN = SIN(TWO * Z)
        CW1 = ONE
        SW1 = ZERO
C
        DO 80 LITLA = 1, IFACA
           DO 60 I0 = LITLA, N, IFCAB
C
C THIS IS THE MAIN CALCULATION OF RADIX 4 TRANSFORMS
C
              I1 = I0 + IFACA
              I2 = I1 + IFACA
              I3 = I2 + IFACA
              XS0 = XREAL(I0) + XREAL(I2)
              XS1 = XREAL(I0) - XREAL(I2)
              YS0 = XIMAG(I0) + XIMAG(I2)
              YS1 = XIMAG(I0) - XIMAG(I2)
              XS2 = XREAL(I1) + XREAL(I3)
              XS3 = XREAL(I1) - XREAL(I3)
              YS2 = XIMAG(I1) + XIMAG(I3)
```

```
                 YS3 = XIMAG(I1) - XIMAG(I3)
                 XREAL(I0) = XS0 + XS2
                 XIMAG(I0) = YS0 + YS2
                 X1 = XS1 + YS3
                 Y1 = YS1 - XS3
                 X2 = XS0 - XS2
                 Y2 = YS0 - YS2
                 X3 = XS1 - YS3
                 Y3 = YS1 + XS3
                 IF (LITLA - 1) 170, 40, 50
40               XREAL(I2) = X1
                 XIMAG(I2) = Y1
                 XREAL(I1) = X2
                 XIMAG(I1) = Y2
                 XREAL(I3) = X3
                 XIMAG(I3) = Y3
                 GO TO 60
C
C MULTIPLY BY TWIDDLE FACTORS IF REQUIRED
C
50               XREAL(I2) = X1 * CW1 + Y1 * SW1
                 XIMAG(I2) = Y1 * CW1 - X1 * SW1
                 XREAL(I1) = X2 * CW2 + Y2 * SW2
                 XIMAG(I1) = Y2 * CW2 - X2 * SW2
                 XREAL(I3) = X3 * CW3 + Y3 * SW3
                 XIMAG(I3) = Y3 * CW3 - X3 * SW3
60            CONTINUE
              IF (LITLA - IFACA) 70, 80, 170
C
C CALCULATE A NEW SET OF TWIDDLE FACTORS
C
70            Z = CW1 * BCOS - SW1 * BSIN + CW1
              SW1 = BCOS * SW1 + BSIN * CW1 + SW1
              TEMPR = ONE5 - HALF * (Z * Z + SW1 * SW1)
              CW1 = Z * TEMPR
              SW1 = SW1 * TEMPR
              CW2 = CW1 * CW1 - SW1 * SW1
              SW2 = TWO * CW1 * SW1
              CW3 = CW1 * CW2 - SW1 * SW2
              SW3 = CW1 * SW2 + CW2 * SW1
80         CONTINUE
C
           IF (IFACA - 1) 120, 120, 90
C
C SET UP THE TRANSFORM SPLIT FOR THE NEXT STAGE
C
90         IFACA = IFACA / 4
           IF (IFACA) 170, 100, 30
C
C THIS IS THE CALCULATION OF A RADIX TWO STAGE
C
100        DO 110 K = 1, N, 2
              TEMPR = XREAL(K) + XREAL(K + 1)
              XREAL(K + 1) = XREAL(K) - XREAL(K + 1)
              XREAL(K) = TEMPR
              TEMPR = XIMAG(K) + XIMAG(K + 1)
              XIMAG(K + 1) = XIMAG(K) - XIMAG(K + 1)
              XIMAG(K) = TEMPR
110        CONTINUE
C
```

```
120     IF (ITYPE) 130, 170, 150
C
C IF THIS WAS AN INVERSE TRANSFORM, CONJUGATE THE RESULT
C
130     DO 140 K = 1, N
           XIMAG(K) = -XIMAG(K)
140     CONTINUE
C
        RETURN
C
C IF THIS WAS A FORWARD TRANSFORM, SCALE THE RESULT
C
150     Z = ONE / FLOAT(N)
        DO 160 K = 1, N
           XREAL(K) = XREAL(K) * Z
           XIMAG(K) = XIMAG(K) * Z
160     CONTINUE
C
170     RETURN
        END
C
C
C ************************************************************************
C
C
        SUBROUTINE SCRAM(XREAL, XIMAG, N, IPOW)
C
C SOURCE:  DONALD M. MONRO, "COMPLEX DISCRETE FAST FOURIER
C          TRANSFORM", APPLIED STATISTICS 24(1975)153-160.
C
        INTEGER L(21), J(21)
        REAL XREAL(N), XIMAG(N)
        J(1) = 1
        II = 1
C
        DO 10 K = 1, IPOW
           L(K) = II
           II = II * 2
10      CONTINUE
C
        KLOOP = 1
        II = 1
        KPOW = IPOW - 1
        ITOP = 2 ** KPOW
        JPOW = IPOW - 2
C
C PROPAGATE THE INITIAL VALUE FOR THE COUNTER FOR THIS LOOP
C
20      DO 30 K = KLOOP, JPOW
           J(K + 1) = J(K)
30      CONTINUE
C
C J20 IS THE BIT-REVERSE OF II
C
        J20 = J(KPOW)
C
        DO 60 I = 1, 2
           IF (II - J20) 40, 50, 50
C
C PAIRWISE INTERCHANGE
C
```

```
40          TEMPR = XREAL(II)
            XREAL(II) = XREAL(J20)
            XREAL(J20) = TEMPR
            TEMPR = XIMAG(II)
            XIMAG(II) = XIMAG(J20)
            XIMAG(J20) = TEMPR
C
C INCREMENT THE INNER LOOP
C
50          J20 = J20 + ITOP
            II = II + 1
60       CONTINUE
C
         KLOOP = KPOW
C
C INCREMENT AND TEST OUTER LOOPS
C
70       J(KLOOP) = J(KLOOP) + L(KLOOP)
         IF (J(KLOOP) - L(KLOOP + 1)) 20, 20, 80
80       KLOOP = KLOOP - 1
         IF (KLOOP) 90, 90, 70
C
90       RETURN
         END
C
C
C **********************************************************************
C
C
         SUBROUTINE TRANSP(IDEV, IMARK, MPOW, M, NPOW, N, NHALF, IP, IPP,
      &  IPAIR, IROWNR, ROWS, BUF)
C
C THIS SUBROUTINE TRANSPOSES A REAL MATRIX OF SIZE N X N (WHERE
C N = 2 ** NPOW AND NPOW IS AN INTEGER GREATER THAN OR EQUAL TO 2)
C USING THE EKLUNDH FAST MATRIX TRANSPOSTION ALGORITHM.
C
         INTEGER IP(MPOW, NHALF), IPP(MPOW, NHALF), IPAIR(MPOW), IROWNR(M)
         REAL ROWS(M, N), BUF(N)
C
         NPASS = NPOW / MPOW
         IF (JMOD(NPOW, MPOW) .NE. 0) NPASS = NPASS + 1
C
         DO 180 IPASS = 1, NPASS
            ISTART = 1 + MPOW * (IPASS - 1)
            IEND = MPOW * IPASS
            IF (IEND .GT. NPOW) IEND = NPOW
            DO 20 I = ISTART, IEND
               J = I - 1
               IDELTA = 2 ** J
               ICOUNT = 1
               ISTOR = JMOD(I - 1, MPOW) + 1
               DO 10 IROW = 1, N
                  IROWP = IROW - 1
                  IROWPP = IROWP + IDELTA
                  IF (IROWPP .GT. N - 1) GOTO 10
                  IJP = JMOD((IROWP / IDELTA), 2)
                  IJPP = JMOD((IROWPP / IDELTA), 2)
                  IF (IJP .EQ. IJPP) GOTO 10
                  ITEMP = IROWP
                  IF (IJP .EQ. 0) ITEMP = ITEMP + IDELTA
                  IF (IJP .EQ. 1) ITEMP = ITEMP - IDELTA
                  IF ((ITEMP - IROWPP) .NE. 0) GOTO 10
                  IP(ISTOR, ICOUNT) = IROWP
                  IPP(ISTOR, ICOUNT) = IROWPP
                  IF (ICOUNT .EQ. NHALF) GOTO 20
                  ICOUNT = ICOUNT + 1
10             CONTINUE
20          CONTINUE
            NSTOR = ISTOR
            DO 40 I = 1, NSTOR
               DO 30 J = 1, NHALF
                  IP(I, J) = IP(I, J) + 1
                  IPP(I, J) = IPP(I, J) + 1
30             CONTINUE
40          CONTINUE
            DO 50 I = 1, NSTOR
               IPAIR(I) = 1
50          CONTINUE
            IDELTA = M ** (IPASS - 1)
            ICOUNT = 1
60          IDUMM = 0
70          IROW = JMOD(ICOUNT - 1, M) + 1
            IRMATR = JMOD(IDELTA * (ICOUNT - 1), N - 1) + 1
            IF (ICOUNT .EQ. N) IRMATR = N
            IROWNR(IROW) = IRMATR
            READ (IDEV, REC = IRMATR) BUF
            DO 80 K = 1, N
               ROWS(IROW, K) = BUF(K)
80          CONTINUE
            ICOUNT = ICOUNT + 1
            IF (IROW .NE. M) GOTO 70
            DO 150 ISTOR = 1, NSTOR
90             J = JMOD(IDELTA * (IPAIR(ISTOR) - 1), NHALF - 1) + 1
               IF (IPAIR(ISTOR) .EQ. NHALF) J = NHALF
               DO 130 K = 1, M
                  IF (IP(ISTOR, J) .NE. IROWNR(K)) GOTO 130
                  KP = K
                  DO 100 L = KP + 1, M
                     IF (IPP(ISTOR, J) .NE. IROWNR(L)) GOTO 100
                     LP = L
                     GOTO 110
100               CONTINUE
110               DO 120 L = 1, NHALF
                     IPL = IP(ISTOR, L)
                     IPPL = IPP(ISTOR, L)
                     STOR = ROWS(KP, IPPL)
                     ROWS(KP, IPPL) = ROWS(LP, IPL)
                     ROWS(LP, IPL) = STOR
120               CONTINUE
                  GOTO 140
130            CONTINUE
140            IPAIR(ISTOR) = IPAIR(ISTOR) + 1
               IF (JMOD(IPAIR(ISTOR) - 1, M / 2) .NE. 0) GOTO 90
150         CONTINUE
            DO 170 IROW = 1, M
               DO 160 I = 1, N
                  BUF(I) = ROWS(IROW, I)
160            CONTINUE
               WRITE (IDEV, REC = IROWNR(IROW)) BUF
```

```
170         CONTINUE
            IF (ICOUNT .LT. N) GOTO 60
180      CONTINUE
C
         RETURN
         END
```

```
C TTRANSP.FOR
C
C THIS PROGRAM TESTS THE FAST MATRIX TRANSPOSITION SUBROUTINE, TRANSP,
C ON REAL MATRICES RANGING IN SIZE FROM 4 X 4 TO 1024 X 1024.
C
      INTEGER IP(10, 512), IPP(10, 512), IPAIR(10), IROWNR(1024)
      REAL ROWS(1024, 1024), BUF(1024)
C
      DO 20 NPOW = 2, 10
         N = 2 ** NPOW
         NHALF = N / 2
         DO 10 MPOW = 1, NPOW
            M = 2 ** MPOW
            CALL TRANTEST(MPOW, M, NPOW, N, NHALF, IP, IPP, IPAIR,
     &         IROWNR, ROWS, BUF)
10       CONTINUE
20    CONTINUE
C
      STOP
      END
C
C
      SUBROUTINE TRANTEST(MPOW, M, NPOW, N, NHALF, IP, IPP, IPAIR,
     & IROWNR, ROWS, BUF)
C
      INTEGER IP(MPOW, NHALF), IPP(MPOW, NHALF), IPAIR(MPOW), IROWNR(M)
      REAL ROWS(M, N), BUF(N)
C
      OPEN (UNIT = 11, ACCESS = 'DIRECT', RECL = N, MAXREC = N,
     &     ASSOCIATE VARIABLE = I11, STATUS = 'SCRATCH')
      OPEN (UNIT = 12, FILE = 'TEST.OUT', STATUS = 'NEW')
C
      DO 20 IROW = 1, N
         DO 10 ICOL = 1, N
            BUF(ICOL) = FLOAT(N * (IROW - 1) + ICOL)
10       CONTINUE
         WRITE (11, REC = IROW) BUF
20    CONTINUE
C
      CALL TRANSP(11, I11, MPOW, M, NPOW, N, NHALF, IP, IPP,
     & IPAIR, IROWNR, ROWS, BUF)
C
      ICOUNT = 0
C
      DO 50 IROW = 1, N
         READ (11, REC = IROW) BUF
         DO 40 ICOL = 1, N
            DIFF = ABS(BUF(ICOL) - FLOAT(N * (ICOL - 1) + IROW))
            IF (DIFF .GT. 1.0E-6) THEN
               WRITE (12, 30) IROW, ICOL, DIFF
30             FORMAT(1X, 2I8, F12.4)
               ICOUNT = ICOUNT + 1
            END IF
40       CONTINUE
50    CONTINUE
C
      WRITE (12, 60) MPOW, NPOW, ICOUNT
      WRITE (6, 60) MPOW, NPOW, ICOUNT
60    FORMAT(1X, //, 5X, 'MPOW = ', I3, 5X, 'NPOW = ', I3, 5X,
     & 'ERROR COUNT = ', I8)
```

```
C
      CLOSE (11, STATUS = 'DELETE')
      RETURN
      END
```

```
C TFFT2D.FOR
C
C THIS PROGRAM TESTS THE ACCURACY OF FFT2D IN PERFORMING BOTH FORWARD
C AND INVERSE FOURIER TRANSFORMS ON ARRAYS OF SIZE 256 X 256.  IT
C CAN BE EASILY MODIFIED TO HANDLE ARRAYS OF SIZE N X N, WHERE N IS A
C POWER OF TWO AND N IS GREATER THAN OR EQUAL TO FOUR.
C
      INTEGER IP(4, 128), IPP(4, 128), IPAIR(4), IROWNR(16), ROWBEG,
     & ROWEND, COLBEG, COLEND
      REAL ROWS(16, 256), XREAL(256), XIMAG(256), BUFR(256), BUFI(256)
      DOUBLE PRECISION DC(256), DS(256), DCC(256), DSS(256), DPI,
     & DFAC1, DFAC2, DARG1, DARG2, DELTAX, DELTAY, DRR, DII, DRFFT,
     & DIFFT, DELR, DELI, FR, FI, FFR, FFI, XK, YL, UM, VN, XX, YY
      COMMON /XXYY /XX, YY
C
      MPOW = 4
      NPOW = 8
      ROWBEG = 128
      ROWEND = 128
      COLBEG = 151
      COLEND = 154
      XX = 1.0D0
      YY = 1.0D0
C
      M = 2 ** MPOW
      N = 2 ** NPOW
      NHALF = N / 2
      DPI = 3.1415926535897932D0
      DFAC1 = (2.0D0 * DPI) / DFLOAT(N)
      DFAC2 = DFLOAT(N - 1) / 2.0D0
C
      OPEN(UNIT = 11, ACCESS = 'DIRECT', FILE = 'REAL.DAT',
     &    RECL = N, MAXREC = N, ASSOCIATE VARIABLE = I11,
     &    STATUS = 'UNKNOWN')
      OPEN(UNIT = 12, ACCESS = 'DIRECT', FILE = 'IMAG.DAT',
     &    RECL = N, MAXREC = N, ASSOCIATE VARIABLE = I12,
     &    STATUS = 'UNKNOWN')
      OPEN(UNIT = 13, FILE = 'TFFT2D.DAT', STATUS = 'UNKNOWN')
C
      DELTAX = XX / DFLOAT(N)
      DELTAY = YY / DFLOAT(N)
C
C GENERATE SAMPLES OF COMPLEX FUNCTION TO BE FOURIER TRANSFORMED; STORE
C IN DISK FILES
C
      DO 20 IROW = 1, N
         L = N - IROW
         YL = (DFLOAT(L) - DFAC2) * DELTAY
         DO 10 ICOL = 1, N
            K = ICOL - 1
            XK = (DFLOAT(K) - DFAC2) * DELTAX
            BUFR(ICOL) = FR(XK, YL)
            BUFI(ICOL) = FI(XK, YL)
10       CONTINUE
         WRITE(11, REC = IROW) BUFR
         WRITE(12, REC = IROW) BUFI
20    CONTINUE
C
C CALCULATE FORWARD DFT USING FFT ALGORITHM
C
```

```
      END
C
      FUNCTION FI(X, Y)
      DOUBLE PRECISION FI, X, Y
C
      FI = 0.0D0
C
      RETURN
      END
```

```
C CONVOLVE.FOR
C
C THIS PROGRAM CONVOLVES TWO COMPLEX ARRAYS (REAL AND IMAGINARY
C PARTS STORED IN SEPARATE FILES) OF SIZE N X N, WHERE N = 512.
C THE PROGRAM CAN BE EASILY MODIFIED TO HANDLE OTHER CASES WHERE
C N IS A POWER OF TWO AND N IS GREATER THAN OR EQUAL TO TWO.
C CONVOLVING TWO 512 X 512 COMPLEX ARRAYS ON A DEC MICROVAX II
C COMPUTER REQUIRES APPROXIMATELY TWO HOURS OF EXECUTION TIME.
C
        INTEGER IP(5, 512), IPP(5, 512), IPAIR(5), IROWNR(32)
        REAL ROWS(32, 1024), XREAL(1024), XIMAG(1024), AR(512), AI(512),
     &  BR(512), BI(512), CR(1024), CI(1024), DR(1024), DI(1024),
     &  ZEROES(1024)
        DOUBLE PRECISION DC(1024), DS(1024), DCC(1024), DSS(1024)
        CHARACTER * 24 DATIN1, DATIN2, DATIN3, DATIN4, DATO1, DATO2
C
        MPOW = 5
        NPOW = 10
        M2 = 2 ** MPOW
        N2 = 2 ** NPOW
        N = N2 / 2
        N2QUAR = N2 / 4
C
C OBTAIN USER INPUTS
C
        WRITE (6, *) 'ENTER REAL, IMAG DATA FILES -- IN APOSTROPHES'
        WRITE (6, *) 'AREAL, AIMAG, BREAL, BIMAG, DREAL, DIMAG'
        READ (5, *) DATIN1, DATIN2, DATIN3, DATIN4, DATO1, DATO2
C
        OPEN (UNIT = 11, ACCESS = 'DIRECT', FILE = DATIN1, RECL = N,
     &       MAXREC = N, ASSOCIATE VARIABLE = I11, STATUS = 'OLD')
        OPEN (UNIT = 12, ACCESS = 'DIRECT', FILE = DATIN2, RECL = N,
     &       MAXREC = N, ASSOCIATE VARIABLE = I12, STATUS = 'OLD')
        OPEN (UNIT = 13, ACCESS = 'DIRECT', FILE = DATIN3, RECL = N,
     &       MAXREC = N, ASSOCIATE VARIABLE = I13, STATUS = 'OLD')
        OPEN (UNIT = 14, ACCESS = 'DIRECT', FILE = DATIN4, RECL = N,
     &       MAXREC = N, ASSOCIATE VARIABLE = I14, STATUS = 'OLD')
C
        OPEN (UNIT = 15, ACCESS = 'DIRECT', RECL = N2, MAXREC = N2,
     &       ASSOCIATE VARIABLE = I15, STATUS = 'SCRATCH')
        OPEN (UNIT = 16, ACCESS = 'DIRECT', RECL = N2, MAXREC = N2,
     &       ASSOCIATE VARIABLE = I16, STATUS = 'SCRATCH')
        OPEN (UNIT = 17, ACCESS = 'DIRECT', FILE = DATO1, RECL = N2,
     &       MAXREC = N2, ASSOCIATE VARIABLE = I17, STATUS = 'UNKNOWN')
        OPEN (UNIT = 18, ACCESS = 'DIRECT', FILE = DATO2, RECL = N2,
     &       MAXREC = N2, ASSOCIATE VARIABLE = I18, STATUS = 'UNKNOWN')
C
        DO 10 I = 1, N2
           CR(I) = 0.0
           CI(I) = 0.0
           DR(I) = 0.0
           DI(I) = 0.0
           ZEROES(I) = 0.0
10      CONTINUE
C
C EXPAND ORIGINAL N X N MATRICES TO SIZE N2 X N2 BY FIRST PLACING
C EACH ORIGINAL IN CENTER OF LARGER N2 X N2 ARRAY; THEN FILL "COLLAR" REGION
C WITH ZEROES
C
        DO 30 IROW = 1, N
```